Class 1 - WPF Introduction

Summary:

- Event-driven Programming
- Development Tools
- Basic Concepts
- XAML Introduction

Sequential Programming

Sequential programming refers to programs that are made up of lines of code that are executed one after the other from beginning to end. When the program is executed it runs without any delay in a sequential manner.

```
Main()
{
   Code line
   Code line
   Code line
}
```

Pseudo-code: sequential programming

Event-driven Programming:

A different programming paradigm is used for interface programming, known as event-driven programming. In this type of programs, there are a series of controls or objects (for example, buttons, text boxes, lists, etc.) and the program is waiting for an event to occur in those controls, such as: clicking a button or pressing a button key. After the occurrence of an event, the procedure associated with the event that occurred is performed. WPF, like almost all languages developed to create graphical interfaces, is based on this programming paradigm.

In this type of programming, the program must return to various events by calling an event handling. The system has to associate an observer (`event listener or observer`) to the various types of events (`event source`), which indicates which method will be executed in response to the generated event (`callback`).

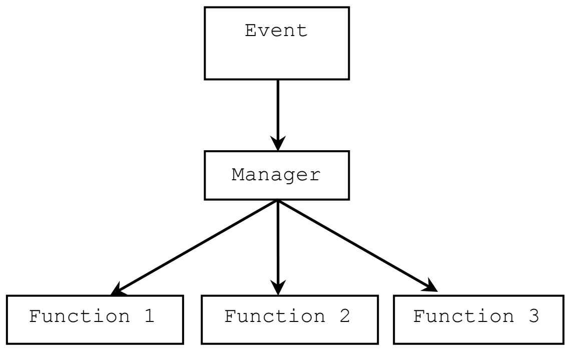

Figure 1-1: Event Handling

```
Main()
{
     Initialization code
     Registering Callbacks
     Main Cycle
}

Callback_1()
{
  Code line
  Code line
  Code line
}

Callback_2()
{
     …
}
```

Figure 1-2: Pseudo-code – Event-driven Programming

1.1. Development Tools / WPF

Windows Presentation Foundation (WPF) is Microsoft's framework for the development of interactive applications combining interfaces, 2D and 3D graphics, documents and multimedia in one tool. WPFs separate the interface and logic layers of programs. The XAML (*Extensible Application Markup Language*) is used for this purpose, an XML-based language that allows the creation and initialization of .NET components.

To develop applications in WPF the main tool used is *Visual Studio*, which is more dedicated to programmers, allowing them to create and edit XAML code.

1.2. First Example

In *Visual Studio*, create a new WPF application (`File->New->Project`). Choose `Visual C# - WPF app (.NET Framework)` as the type of application and choose a directory and a name for the project.

Explore the code created by the application, note the existence of two code files with the `.cs` extension and two additional files with the `.xaml` extension that describe the interface.

The events are already pre-defined in Visual C#, with a large number of events that can be associated with code.
The first event that we will explore is the click on the main window.

In the interface view (by clicking on the `.xaml` file and choosing `View Designer`), analyse the `Windows` properties (be careful to select the `window` control and not the `grid` control) and select the events `icon` ⚡ that allows you to view the events associated with the selected control.

In the list of events, double-click the `MouseLeftButtonDown` option and observe what happens at the code level.

Add the following code to the function that is automatically created which allows you to write a line in the Visual C# output window:

```
Console.WriteLine ("Hello World");
```

Run the code and observe what happens in the `Output` window whenever you click on the window control. Check that you are viewing the *Visual Studio* `Output` window while running the program, as the write command on the console writes to that specific window. To activate the window, use the option `View->Output`.

1.3. Other Events in Visual C#

Modify the code to respond to other types of events by placing a specific message for each event in the output window. Explore and choose other four events (associating a different message for each one) among the following (`Activated, MouseMove, SizeChanged, MouseDoubleclick, MouseDown, Closed, Loaded, DragEnter, Drop, etc.`). What events do they correspond to? Note that for some events it may be necessary to modify window attributes (e.g. `AllowDrop`)

1.4. Associating Several Controls to the Same Event

It is possible to associate several controls with the same event. If you wish, you can remove the code added in the last paragraph but please note that when removing the code associated with an event, it is necessary to remove the line in the corresponding `.xaml` file on which the event is defined.

Add a button to the `grid` using the `toolbox`. Change the text to `button1` by changing the `content` attribute in the xaml or interface. Associate the click event with the following code, which shows a message in a dialog box:

```
MessageBox.Show ("Hello World");
```

To define the event, you can simply double-click the button (the click event is defined by default). When associating the click event to button1, *Visual Studio* automatically creates the following code:

```
private void button1_Click (object sender, RoutedEventArgs e)
{
}
```

Visual Studio created a C# subroutine named `Button1_Click` that is invoked whenever button 1 is clicked.

Place a new button (`button2`) that should be associated with the same subroutine. To do so, simply associate the function already created using the properties of the second button to the second button click event.

You can add a breakpoint (F9 or `Debug->Toggle Breakpoint`) in the function to verify that it is invoked by clicking on any of the buttons.

The two events "click on button 1" and "click on button2" are now associated with the same subroutine. Note, however, that when you click on button 2, it jumps to the code associated with button 1 and it is no longer possible to have a specific subroutine for button 2 since it is already associated with another subroutine.

1.5. Events Arguments

The arguments for the `Button1_click` function that we saw earlier are: `object sender` and `RoutedEventArgs e`.. The `sender` is a reference to the control/object that triggered the event (the button that was pressed, for example). The variable `e` contains information related to the event, for example the button or the position of the mouse on the monitor in case of clicking on a button.

To illustrate the use of arguments, let's use another event: `MouseMove`.

Start by removing all the code from the previous paragraph until you remain with only one button. Associate the `MouseMove` event to the button. Use the second argument of the function (of type `MouseEventArg`) to determine which button was used. To achieve this result, you can use the following code:

```
if (e.RightButton == MouseButtonState.Pressed)
        Console.WriteLine ("Right Button");
```

Run the code and see what happens when you click the mouse buttons.

Add a function to detect a double click of the mouse in the window and indicate the position where the click occurred. To achieve this result, you can use the code below:

```
Point p = e.GetPosition(this);
double xPos = p.X;
double yPos = p.Y;
Console.WriteLine("The X Position is " + xPos + " The Y Position
is " + yPos);
```

Test the code again. What happens? Using this program, try to determine which are the coordinates that are used at the `window` level trying to determine the maximum and minimum limits in pixels and check where that dimension is defined in the xaml.

1.6. Animation

Modify the code so that the message is not displayed but to change the `content` of the button by clicking on it (for example, I change the title of button1 to Hello World).

Now include the animation library in the file:

```
using System.Windows.Media.Animation
```

Add in the click function the following code:

```
    SolidColorBrush init_color = new SolidColorBrush(Colors.DarkGray);
    this.Background = init_color;
    ColorAnimation colorAnim = new ColorAnimation();
    colorAnim.Duration = new
Duration(TimeSpan.FromMilliseconds(2000));
    colorAnim.To = Colors.LightGray;
    init_color.BeginAnimation(SolidColorBrush.ColorProperty,
colorAnim);
```

Modify the program so that the background of the window matches the colour of the beginning of the animation. Also modify some parameters of the animation (initial color, final colour, time, etc ...).

1.7. Using XAML

Use the following code to add a new button directly to the XAML file instead of using the *Visual Studio* Designer.

```
  <Button Content="XAML button" Margin="24,195,22,20" Name="button2"
FontSize="25"/>
```

Run the code and see how it behaves if you change the window size.

1.8. Data Binding Simple Example

One of the advantages of XAML is the possibility of associating information with controls (`data binding`). Add a `slider` and a `textbox` to the interface. In the XAML associated with the slider and textbox, associate the following code, keeping the rest of the code unchanged:

```
    <Slider x:Name ="SliderName"/>
    <TextBox Text="{Binding Value, ElementName= SliderName}"/>
```

This code associates a unique name to the slider (directive x: name) so that you can later refer to it in the code. The content of the textbox (text) is then associated with that control. Modify the code so that the returned value is between 0 and 100. Investigate the `StringFormat`[1] property that allows you to format the appearance of the data. See formatting numeric data for `string`[2] and format the example to show only 2 decimal places.

1.9. A Small Program to Monitor the Mouse in a Window

Using all the code and the examples that were introduced in this class, create a small program that allows you to monitor the mouse in a window. The program should create an empty `window` and perform the following operations:

• Count the number of mouse clicks, distinguishing between the right button and the left button;
• Indicate on the console the x,y position where each mouse click occurs.
• Add the entire movement that the mouse performs inside the window.
• When closing, view the results of the monitoring using the code:

```
MessageBox.Show("Text to display”);
```

Once this program is complete, modify it by adding the necessary controls to view the values in the window as you interact with it.

[1]https://msdn.microsoft.com/en-us/library/system.windows.data.bindingbase.stringformat%28v=vs.110%29.aspx
[2]https://msdn.microsoft.com/en-us/library/0c899ak8(v=vs.110).aspx